MySQL Database Dump

We store all student state in MySQL. This includes registration information, enrollment, and certificate status.

General conventions:

- All strings are stored as UTF-8.
- All datetimes are stored in UTC.

All the tables will be described below, first in summary form with field types and constraints, and then with a detailed explanation of each field. For those not familiar with the MySQL schema terminology in the table summaries:

Type

This is the kind of data it is, along with the size of the field. When a numeric field has a length specified, it just means that's how many digits we want displayed -- it has no affect on the number of bytes used.

- `int` = 4 byte integer.
- `smallint` = 2 byte integer, sometimes used for enumerated values.
- `tinyint` = 1 byte integer, but usually just used to indicate a boolean field with 0 = False and 1 = True.
- `varchar` = String, typically short and indexable. The length is the number of chars, not bytes (so unicode friendly).
- `longtext` = A long block of text, usually not indexed.
- `date` = Date
- `datetime` = Datetime in UTC, precision in seconds.

Null

- `YES` = NULL values are allowed.
- `NO` = NULL values are not allowed. That being said, Django often just places blank strings instead of NULL when it wants to indicate a text value is optional. This is used more for numeric and date fields.

Key

- `PRI` = Primary key for the table, usually named `id`, unique
- `UNI` = Unique
- `MUL` = Indexed for fast lookup, but the same value can appear multiple times. A Unique index that allows NULLs can also show up as MUL.

`auth_user`

The `auth_user` table is built into the Django web framework that we use. It holds generic information necessary for basic login and permissions information. It has the following fields:

```
+------------------------------+--------------+------+-----+
| Field                        | Type         | Null | Key |
+------------------------------+--------------+------+-----+
| id                           | int(11)      | NO   | PRI |
| username                     | varchar(30)  | NO   | UNI |
| first_name                   | varchar(30)  | NO   |     | # Never used
| last_name                    | varchar(30)  | NO   |     | # Never used
| email                        | varchar(75)  | NO   | UNI |
| password                     | varchar(128) | NO   |     |
| is_staff                     | tinyint(1)   | NO   |     |
| is_active                    | tinyint(1)   | NO   |     |
| is_superuser                 | tinyint(1)   | NO   |     |
| last_login                   | datetime     | NO   |     |
| date_joined                  | datetime     | NO   |     |
| status                       | varchar(2)   | NO   |     | # No longer used
| email_key                    | varchar(32)  | YES  |     | # No longer used
| avatar_type                  | varchar(1)   | NO   |     | # No longer used
| country                      | varchar(2)   | NO   |     | # No longer used
| show_country                 | tinyint(1)   | NO   |     | # No longer used
| date_of_birth                | date         | YES  |     | # No longer used
| interesting_tags             | longtext     | NO   |     | # No longer used
| ignored_tags                 | longtext     | NO   |     | # No longer used
| email_tag_filter_strategy    | smallint(6)  | NO   |     | # No longer used
| display_tag_filter_strategy  | smallint(6)  | NO   |     | # No longer used
| consecutive_days_visit_count | int(11)      | NO   |     | # No longer used
+------------------------------+--------------+------+-----+
```

`id`

Primary key, and the value typically used in URLs that reference the user. A user has the same value for `id` here as they do in the MongoDB database's users collection. Foreign keys referencing `auth_user.id` will often be named `user_id`, but are sometimes named `student_id`.

`username`

The unique username for a user in our system. It may contain alphanumeric, _, @, +, . and - characters. The username is the only information that the students give about themselves that we currently expose to other students. We have never allowed people to change their usernames so far, but that's not something we guarantee going forward.

`first_name`

Not used; we store a user's full name in `auth_userprofile.name` instead.

`last_name`

Not used; we store a user's full name in `auth_userprofile.name` instead.

`email`

Their email address. While Django by default makes this optional, we make it required, since it's the primary mechanism through which people log in. Must be unique to each user. Never shown to other users.

`password`

A hashed version of the user's password. Depending on when the password was last set, this will either be a SHA1 hash or PBKDF2 with SHA256 (Django 1.3 uses the former and 1.4 the latter).

`is_staff`

This value is 1 if the user is a staff member *of edX* with corresponding elevated privileges that cut across courses. It does not indicate that the person is a member of the course staff for any given course. Generally, users with this flag set to 1 are either edX program managers responsible for course delivery, or edX developers who need access for testing and debugging purposes. People who have `is_staff = 1` get instructor privileges on all courses, along with having additional debug information show up in the instructor tab.

Note that this designation has no bearing with a user's role in the forums, and confers no elevated privileges there.

Most users have a 0 for this value.

`is_active`

This value is 1 if the user has clicked on the activation link that was sent to them when they created their account, and 0 otherwise. Users who have `is_active = 0` generally cannot log into the system. However, when users first create their account, they are automatically logged in even though they are not active. This is to let them experience the site immediately without having to check their email. They just get a little banner at the top of their dashboard reminding them to check their email and activate their account when they have time. If they log out, they won't be able to log back in again until they've activated. However, because our sessions last a long time, it is theoretically possible for someone to use the site as a student for days without being "active".

Once `is_active` is set to 1, the only circumstance where it would be set back to 0 would be if we decide to ban the user (which is very rare, manual operation).

is_superuser

Value is 1 if the user has admin privileges. Only the earliest developers of the system have this set to 1, and it's no longer really used in the codebase. Set to 0 for almost everybody.

last_login

A datetime of the user's last login. Should not be used as a proxy for activity, since people can use the site all the time and go days between logging in and out.

date_joined

Date that the account was created (NOT when it was activated).

Other Fields

All the following fields were added by an application called Askbot, a discussion forum package that is no longer part of the system:

- `status`
- `email_key`
- `avatar_type`
- `country`
- `show_country`
- `date_of_birth`
- `interesting_tags`
- `ignored_tags`
- `email_tag_filter_strategy`
- `display_tag_filter_strategy`
- `consecutive_days_visit_count`

Only users who were part of the prototype 6.002x course run in the Spring of 2012 would have any information in these fields. Even with those users, most of this information was never collected. Only the fields that are automatically generated have any values in them, such as tag settings.

These fields are completely unrelated to the discussion forums we currently use.

auth_userprofile

The `auth_userprofile` table is mostly used to store user demographic information collected during the signup process. We also use it to store certain additional metadata relating to certificates. Every row in this table corresponds to one row in `auth_user`.

```
+--------------------+--------------+------+-----+
| Field              | Type         | Null | Key |
+--------------------+--------------+------+-----+
| id                 | int(11)      | NO   | PRI |
| user_id            | int(11)      | NO   | UNI |
| name               | varchar(255) | NO   | MUL |
| language           | varchar(255) | NO   | MUL | # Prototype course users only
| location           | varchar(255) | NO   | MUL | # Prototype course users only
| meta               | longtext     | NO   |     |
| courseware         | varchar(255) | NO   |     | # No longer used
| gender             | varchar(6)   | YES  | MUL | # Only users signed up after prototype
| mailing_address    | longtext     | YES  |     | # Only users signed up after prototype
| year_of_birth      | int(11)      | YES  | MUL | # Only users signed up after prototype
| level_of_education | varchar(6)   | YES  | MUL | # Only users signed up after prototype
| goals              | longtext     | YES  |     | # Only users signed up after prototype
| allow_certificate  | tinyint(1)   | NO   |     |
+--------------------+--------------+------+-----+
```

There is an important split between information gathered for the students who signed up during the MITx prototype phase and took 6.002x's first offering in the spring of 2012, and those that signed up afterwards. The first non-prototype user has a `auth_userprofile.user_id` of `156633` and an `auth_userprofile.id` of `155156`.

`id`

Primary key, not referenced anywhere else.

`user_id`

A foreign key that maps to `auth_user.id`.

`name`

String for a user's full name. We make no constraints on language or breakdown into first/last name. The names are never shown to other students. Foreign students usually enter a romanized version of their names, but not always.

It used to be our policy to require manual approval of name changes to guard the integrity of the certificates. Students would submit a name change request and someone from the team would approve or reject as appropriate. Later, we decided to allow the name changes to take place automatically, but to log previous names in the `meta` field.

`language`

User's preferred language, asked during the sign up process for the 6.002x prototype course given in the Spring of 2012. This information stopped being collected after the transition from MITx to edX happened, but we never removed the values from our first group of students. Sometimes written in those languages.

`location`

User's location, asked during the sign up process for the 6.002x prototype course given in the Spring of 2012. We weren't specific, so people tended to put the city they were in, though some just specified their country and some got as specific as their street address. Again, sometimes romanized and sometimes written in their native language. Like `language`, we stopped collecting this field when we transitioned from MITx to edX, so it's only available for our first batch of students.

`meta`

An optional, freeform text field that stores JSON data. This was a hack to allow us to associate arbitrary metadata with a user. An example of the JSON that can be stored here is:

```
{
  "old_names" : [["David Ormsbee", "I want to add my middle name as well.", "2012-11-15T17:28:12.658126"],
                 ["Dave Ormsbee", "Dave's too informal for a certificate.", "2013-02-07T11:15:46.524331"]],
  "old_emails" : [["dormsbee@mitx.mit.edu", "2012-10-18T15:21:41.916389"]],
  "6002x_exit_response" : {
    "rating": ["6"],
    "teach_ee": ["I do not teach EE."],
    "improvement_textbook": ["I'd like to get the full PDF."],
    "future_offerings": ["true"],
    "university_comparison": ["This course was <strong>on the same level</strong> as the university class."],
    "improvement_lectures": ["More PowerPoint!"],
    "highest_degree": ["Bachelor's degree."],
    "future_classes": ["true"],
    "future_updates": ["true"],
    "favorite_parts": ["Releases, bug fixes, and askbot."]
  }
}
```

The following are details about this metadata. Please note that the fields described below are found as JSON attributes *inside* the `meta` field, and are *not* separate database fields of their own.

`old_names`

A list of the previous names this user had, and the timestamps at which they submitted a request to change those names. These name change request submissions used to require a staff member to approve it before the name change took effect. This is no longer the case, though we still record their previous names.

Note that the value stored for each entry is the name they had, not the name they requested to get changed to. People often changed their names as the time for certificate generation approached, to replace nicknames with their actual names or correct spelling/punctuation errors.

The timestamps are UTC, like all datetimes stored in our system.

`old_emails`

A list of previous emails this user had, with timestamps of when they changed them, in a format similar to `old_names`. There was never an approval process for this.

The timestamps are UTC, like all datetimes stored in our system.

`6002x_exit_response`

Answers to a survey that was sent to students after the prototype 6.002x course in the Spring of 2012. The questions and number of questions were randomly selected to measure how much survey length affected response rate. Only students from this course have this field.

`courseware`

This can be ignored. At one point, it was part of a way to do A/B tests, but it has not been used for anything meaningful since the conclusion of the prototype course in the spring of 2012.

`gender`

Dropdown field collected during student signup. We only started collecting this information after the transition from MITx to edX, so prototype course students will have NULL for this field.

Possible values are:

- `NULL` = This student signed up during the 6.002x prototype course, before this information was collected
- `''` (blank) = User did not specify gender.
- `'f'` = Female
- `'m'` = Male
- `'o'` = Other

`mailing_address`

Text field collected during student signup. We only started collecting this information after the transition from MITx to edX, so prototype course students will have NULL for this field. Students who elected not to enter anything will have a blank string.

`year_of_birth`

Dropdown field collected during student signup. We only started collecting this information after the transition from MITx to edX, so prototype course students will have NULL for this field. Students who decided not to fill this in will also have NULL.

`level_of_education`

Dropdown field collected during student signup. We only started collecting this information after the transition from MITx to edX, so prototype course students will have NULL for this field.

Possible values are:

- `NULL` = This student signed up during the 6.002x prototype course, before this information was collected
- `''` (blank) = User did not specify level of education.

- `'p_se'` = Doctorate in science or engineering
- `'p_oth'` = Doctorate in another field
- `'m'` = Master's or professional degree
- `'b'` = Bachelor's degree
- `'hs'` = Secondary/high school
- `'jhs'` = Junior secondary/junior high/middle school
- `'el'` = Elementary/primary school
- `'none'` = None
- `'other'` = Other

`goals`

Text field collected during student signup in response to the prompt, "Goals in signing up for edX". We only started collecting this information after the transition from MITx to edX, so prototype course students will have NULL for this field. Students who elected not to enter anything will have a blank string.

`allow_certificate`

Set to `1` for most students. This field is set to `0` if log analysis has revealed that this student is accessing our site from a country that the US has an embargo against. At this time, we do not issue certificates to students from those countries.

`certificates_generatedcertificate`

```
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| id            | int(11)      | NO   | PRI | NULL    | auto_increment |
| user_id       | int(11)      | NO   | MUL | NULL    |                |
| download_url  | varchar(128) | NO   |     | NULL    |                |
| grade         | varchar(5)   | NO   |     | NULL    |                |
| course_id     | varchar(255) | NO   | MUL | NULL    |                |
| key           | varchar(32)  | NO   |     | NULL    |                |
| distinction   | tinyint(1)   | NO   |     | NULL    |                |
| status        | varchar(32)  | NO   |     | NULL    |                |
| verify_uuid   | varchar(32)  | NO   |     | NULL    |                |
| download_uuid | varchar(32)  | NO   |     | NULL    |                |
| name          | varchar(255) | NO   |     | NULL    |                |
| created_date  | datetime     | NO   |     | NULL    |                |
| modified_date | datetime     | NO   |     | NULL    |                |
| error_reason  | varchar(512) | NO   |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
```

The generatedcertificate table tracks certificate state for students who have been graded after a course completes. Currently the table is only populated when a course ends and a script is run to grade students who have completed the course.

`user_id` and `course_id`

The table is indexed by user and course

`status`

Status may be one of these states

```
unavailable = 'unavailable'
generating = 'generating'
regenerating = 'regenerating'
deleting = 'deleting'
deleted = 'deleted'
downloadable = 'downloadable'
notpassing = 'notpassing'
restricted = 'restricted'
error = 'error'
```

After a course has been graded and certificates have been issued status will be one of:

```
    downloadable = 'downloadable'
    notpassing = 'notpassing'
    restricted = 'restricted'
```

If the status is `downloadable` then the student passed the course and there will be a certificate available for download.

`download_url`, `verify_uuid`, `download_uuid`

The two uuids are what uniquely identify the download url and the url used to download the certificate The `download_uuid` has the full URL to the certificate

`distinction`

This was used for letters of distinction for 188.1x and is not being used for any current courses

`name`

This field records the name of the student that was set at the time the student was graded and the certificate was generated.

`grade`

The grade of the student recorded at the time the certificate was generated. This may be different than the current grade since grading is only done once for a course when it ends.

`courseware_studentmodule`

The `courseware_studentmodule` table holds all courseware state for a given user. Every student has a separate row for every piece of content in the course, making this by far our largest table.

```
+-------------+--------------+------+-----+
| Field       | Type         | Null | Key |
+-------------+--------------+------+-----+
| id          | int(11)      | NO   | PRI |
| module_type | varchar(32)  | NO   | MUL |
| module_id   | varchar(255) | NO   | MUL |
| student_id  | int(11)      | NO   | MUL |
| state       | longtext     | YES  |     |
| grade       | double       | YES  | MUL | # problem, selfassessment, and combinedopenended use this
| created     | datetime     | NO   | MUL |
| modified    | datetime     | NO   | MUL |
| max_grade   | double       | YES  |     | # problem, selfassessment, and combinedopenended use this
| done        | varchar(8)   | NO   | MUL | # ignore this
| course_id   | varchar(255) | NO   | MUL |
+-------------+--------------+------+-----+
```

A Note About Modules

It's important to understand what "modules" are in the context of our system, as the terminology can be confusing. For the conventions of this table and many parts of our code, a "module" is a content piece that appears in the courseware. This can be nearly anything that appears when users are in the courseware tab: a video, a piece of HTML, a problem, etc. Modules can also be collections of other modules, such as sequences, verticals (modules stacked together on the same page), weeks, chapters, etc. In fact, the course itself is a top level module that contains all the other contents of the course as children. You can imagine the entire course as a tree with modules at every node.

Modules can store state, but whether and how they do so is up to the implemenation for that particular kind of module. When a user loads page, we look up all the modules they need to render in order to display it, and then we ask the database to look up state for those modules for that user. If there is corresponding entry for that user for a given module, we create a new row and set the state to an empty JSON dictionary.

`id`

Primary key. Rarely used though, since most lookups on this table are searches on the three tuple of `(course_id, student_id, module_id)`.

`module_type`

Broad category of what kind of module this is. Possible values are:

- `chapter` = The top level categories for a course. Each of these is usually labeled as a Week in the courseware, but this is just convention.
- `combinedopenended` = A new module type developed for grading open ended questions via self assessment, peer assessment, and machine learning.
- `conditional` = A new module type recently developed for 8.02x, this allows you to prevent access to certain parts of the courseware if other parts have not been completed first.
- `course` = The top level course module of which all course content is descended.
- `problem` = A problem that the user can submit solutions for. We have many different varieties.
- `problemset` = A collection of problems and supplementary materials, typically used for homeworks and rendered as a horizontal icon bar in the courseware. Use is inconsistent, and some courses use a `sequential` instead.
- `selfassessment` = Self assessment problems. An early test of the open ended grading system that is not in widespread use yet. Recently deprecated in favor of `combinedopenended`.
- `sequential` = A collection of videos, problems, and other materials, rendered as a horizontal icon bar in the courseware.
- `timelimit` = A special module that records the time you start working on a piece of courseware and enforces time limits, used for Pearson exams. This hasn't been completely generalized yet, so is not available for widespread use.
- `videosequence` = A collection of videos, exercise problems, and other materials, rendered as a horizontal icon bar in the courseware. Use is inconsistent, and some courses use a `sequential` instead.

There's been substantial muddling of our container types, particularly between sequentials, problemsets, and videosequences. In the beginning we only had sequentials, and these ended up being used primarily for two purposes: creating a sequence of lecture videos and exercises for instruction, and creating homework problem sets. The `problemset` and `videosequence` types were created with the hope that our system would have a better semantic understanding of what a sequence actually represented, and could at a later point choose to render them differently to the user if it was appropriate. Due to a variety of reasons, migration over to this has been spotty. They all render the same way at the moment.

`module_id`

Unique ID for a distinct piece of content in a course, these are recorded as URLs of the form `i4x://{org}/{course_num}/{module_type}/{module_name}` (ex: `i4x://MITx/3.091x/problemset/Sample_Problems`). Having URLs of this form allows us to give content a canonical representation even as we are in a state of transition between backend data stores. A breakdown of the parts of a `module_id`:

- `i4x://` = Just a convention we ran with (we had i4x.org but we never ended up using it).
- `org` (ex: `MITx`) = The organization part of the ID, indicating what organization created this piece of content.
- `course_num` (ex: `3.091x`) = The course number this content was created for. Note that there is no run information here, so you can't know what runs of the course this content is being used for from the `module_id` alone; you have to look at the `courseware_studentmodule.course_id` field.
- `module_type` = The module type, same value as what's in the `courseware_studentmodule.module_type` field.
- `module_name` = The name given for this module by the content creators. If the module was not named, the system will generate a name based on the type and a hash of its contents (ex: `selfassessment_03c483062389`).

`student_id`

A reference to `auth_user.id`, this is the student that this module state row belongs to.

`state`

This is a JSON text field where different module types are free to store their state however they wish.

Container Modules: `course`, `chapter`, `problemset`, `sequential`, `videosequence`

The state for all of these is a JSON dictionary indicating the user's last known position within this container. This is 1-indexed, not 0-indexed, mostly because it went out that way at one point and we didn't want to later break saved navigation state for users.

Example: `{"position" : 3}` = When this user last interacted with this course/chapter/etc., they had clicked on the third child element. Note that the position is a simple index and not a `module_id`, so if you rearranged the order of the contents, it would not be smart enough to accomodate the changes and would point users to the wrong place.

The hierarchy goes: `course > chapter > (problemset | sequential | videosequence)`

`combinedopenended`

TODO: More details to come.

`conditional`

Conditionals don't actually store any state, so this value is always an empty JSON dictionary (`'{}'`). We should probably remove these entries altogether.

`problem`

There are many kinds of problems supported by the system, and they all have different state requirements. Note that one problem can have many different response fields. If a problem generates a random circuit and asks five questions about it, then all of that is stored in one row in `courseware_studentmodule`.

TODO: Write out different problem types and their state.

`selfassessment`

TODO: More details to come.

`timelimit`

This very uncommon type was only used in one Pearson exam for one course, and the format may change significantly in the future. It is currently a JSON dictionary with fields:

- `beginning_at` = UTC time as measured in seconds since UNIX epoch representing when the exam was started.
- `ending_at` = UTC time as measured in seconds since UNIX epoch representing the time the exam will close.
- `accomodation_codes` (optional) = Sometimes students are given more time for accessibility reasons. The codes are:
  - `NONE` = 'No Time Accommodation'
  - `ADDHALFTIME` = Extra Time - 1 1/2 Time
  - `ADD30MIN` = Extra Time - 30 Minutes
  - `DOUBLE` = Extra Time - Double Time
  - `TESTING` = Extra Time -- Large amount for testing purposes

`grade`

Floating point value indicating the total unweighted grade for this problem that the student has scored. Basically how many responses they got right within the problem.

Only `problem` and `selfassessment` types use this field. All other modules set this to `NULL`. Due to a quirk in how rendering is done, `grade` can also be `NULL` for a tenth of a second or so the first time that a user loads a problem. The initial load will trigger two writes, the first of which will set the `grade` to `NULL`, and the second of which will set it to `0`.

`created`

Datetime when this row was created (i.e. when the student first accessed this piece of content).

`modified`

Datetime when we last updated this row. Set to be equal to `created` at first. A change in `modified` implies that there was a state change, usually in response to a user action like saving or submitting a problem, or clicking on a navigational element that records its state. However it can also be triggered if the module writes multiple times on its first load, like problems do (see note in `grade`).

`max_grade`

Floating point value indicating the total possible unweighted grade for this problem, or basically the number of responses that are in this problem. Though in practice it's the same for every entry with the same `module_id`, it is technically possible for it to be anything. The problems are dynamic enough where you could create a random number of responses if you wanted. This a bad idea and will probably cause grading errors, but it is possible.

Another way in which `max_grade` can differ between entries with the same `module_id` is if the problem was modified after the `max_grade` was written and the user never went back to the problem after it was updated. This might happen if a member of the course staff puts out a problem with five parts, realizes that the last part doesn't make sense, and decides to remove it. People who saw and answered it when it had five parts and never came back to it after the changes had been made will have a `max_grade` of `5`, while people who saw it later will have a `max_grade` of `4`.

These complexities in our grading system are a high priority target for refactoring in the near future.

Only `problem` and `selfassessment` types use this field. All other modules set this to `NULL`.

`done`

Ignore this field. It was supposed to be an indication whether something was finished, but was never properly used and is just `'na'` in every row.

`course_id`

The course that this row applies to, represented in the form org/course/run (ex: `MITx/6.002x/2012_Fall`). The same course content (same `module_id`) can be used in different courses, and a student's state needs to be tracked separately for each course.

`student_courseenrollment`

A row in this table represents a student's enrollment for a particular course run. If they decide to unenroll in the course, we delete their entry in this table, but we still leave all their state in `courseware_studentmodule` untouched.

`id`

Primary key.

`user_id`

Student's ID in `auth_user.id`

`course_id`

The ID of the course run they're enrolling in (e.g. `MITx/6.002x/2012_Fall`). You can get this from the URL when you're viewing courseware on your browser.

`created`

Datetime of enrollment, UTC.